PROSOFT
Box 839
No. Hollywood, Ca. 91603
(213) 764-3131

Robert W. Brown
780 Manx Ave.
Campbell, CA 95008 U.S.A.

FASTER 1.0

CONGRATULATIONS! You're about to learn how to use a remarkably simple, yet sophisticated software (yes, software) speedup tool for BASIC programs. You don't need to be a skilled programmer to use FASTER, you don't need to be the author of the programs you want to improve. You do need to read this documentation carefully, especially if you want to speed up programs you didn't write yourself. And you need a copy of FASTER, and, of course, a TRS-80.

Since you undoubtedly want to try FASTER on some pet programs, let's get started. Please read as much of this documentation as your patience will permit, then select a relatively simple BASIC program for practice. If you have any programs that you wrote yourself, start with one of them; otherwise, pick something whose current run speed is familiar to you.

WHAT IS "FASTER"?

FASTER is a machine-language utility program for the TRS-80 Model I Level II tape or disk system. It is used to help you speed up the execution speed of most BASIC programs. To do this, a BASIC program is run while FASTER is active in the computer. FASTER counts every reference to every variable used during execution of the BASIC program. Then, at your command, it displays these variables in descending order of their usage. This display can be given on the video or a printer, and lets you add or change a few (often just one) lines of code in the BASIC program. That simple change generally speeds up the program by 10 - 50%! Very simple, very effective.

WHY FASTER IS EFFECTIVE

It's well known that the BASIC interpreter searches a table each time a variable is referenced in a program. The search starts at the beginning of the table, goes to the end, and allocates the variable at the end of the table if it isn't found by then. To make a program faster, its most-used variables should be placed at the beginning of the table. ("Most used" refers to the number of times a variable is encountered during execution, not to the number of times it occurs as a word in the program.) Unfortunately, it's just about impossible for a person to figure this out by examining a BASIC program, and even a relatively small error (allocating a variable a little late) shows up in the form of slower-running programs.

FASTER counts these variable references for you, making it very easy to ensure that the table will be in the best sequence for this particular BASIC program in the future. Further, FASTER only has to be used once for each program you want to speed up. It doesn't have to be in memory each time you run your programs, so once you've optimized a BASIC program, you just save it and run it as you always did in the past... only it runs faster.

INSTALLING FASTER

Level II Tape Systems

1. Power on and answer "MEMORY SIZE" as you normally do.
2. Load any SYSTEM programs you must have when running the BASIC program you plan to speed up.
3. Ready the distribution cassette (it is a SYSTEM program).
4. Type: SYSTEM (and press <ENTER>).
5. Type: FASTER (and press <ENTER>).
6. If the load is successful, then reply to the next "?" prompt by typing a slash ("/") and pressing <ENTER>. If you have problems loading the tape, try other volume settings, cleaning the heads, etc. If these don't work, there is a second copy on side "A", and two more copies on side "B". If none of them can be read, return the original tape to us within 10 days of purchase, with proof of purchase, for a free replacement. Tapes that become unreadable in normal usage will be replaced for a $5.00 handling fee (the original PROSOFT tape must be returned in all cases).
7. The program is now activated and ready for use. All disk-related procedures below may be ignored.

Disk Systems

If you received the program on disk, just copy "FASTER/CMD" to a disk of your own, and keep the PROSOFT disk for backup and possible sending back to us should updates be required.

If you received the program on tape, you can transfer it to disk as follows:

1. Hold down the <BREAK> key and power on or RESET. This will put you into Level II.
2. Hit <ENTER> to reply to the "MEMORY SIZE" question.
3. Ready the distribution cassettte.
4. Type: SYSTEM (and press <ENTER>).
5. Type: FASTER (and press <ENTER>).
6. If you have problems loading the tape, see step 6 in "Level II", above (but don't type "/").
7. Following a successful load, the "?" prompt will re-appear.
8. Hold down the <ENTER> key and re-boot from disk, ending up in DOS (the <ENTER> key bypasses any AUTO procedures).
9. Make sure a writeable diskette containing at least 2 grans of free space is in a disk drive.
10. Use the "DUMP" command to transfer the program to disk:

```
TRSDOS:    DUMP FASTER/CMD (START=X'5200',END=X'583F',TRA=X'5200')
NEWDOS/80: DUMP FASTER/CMD,5200H,583FH,5200H
```

11. Make at least one disk backup of the program.

Activating FASTER (Disk Systems)

To use FASTER in a disk system, load any other necessary machine language programs first. If you're short on memory, try not to load anything that you can do without (such as lower-case drivers, spoolers, and keyboard helpers). If you don't have to load any other machine languge programs, or if all of them set the HIMEM pointer (at X'4049'), then activate FASTER the way you would activate BASIC (FASTER replaces the BASIC command). If your other machine language programs don't update HIMEM, then you can tell FASTER where to start. It is self-relocating and generally compatible with

other machine language programs. If you do have to tell it where to load, take the normal HIMEM or MEM SIZE, and subtract 1250 from it. When you activate FASTER, give it the resulting address, in decimal.

These are the possible formats for activating FASTER:

```
FASTER
FASTER sssss
FASTER sssss,f
FASTER f
FASTER f,mmmmm
FASTER sssss,f,mmmmm
```

'sssss' is the starting address (in decimal) FASTER should use if you have to tell it where to start. If omitted, the value at HIMEM is used as the ending address for FASTER. Omit 'sssss' if you don't have to use it, but if you are using other high-memory machine language programs that don't update HIMEM, (or you aren't sure), then supply a value of 'sssss' that is low enough to leave room for FASTER's 1,250 bytes of code.

'f' is the number of files you will be using. This only applies to NEWDOS. If you are running TRSDOS, you will be asked to enter the number of files just as though you had typed "BASIC".

'mmmmm' is the memory size (in decimal) that should be set for BASIC. This only applies to NEWDOS. If you are running TRSDOS, you will be asked to enter the memory size just as though you had typed "BASIC". (With TRSDOS or NEWDOS, you can normally omit the MEM SIZE, since FASTER sets it.)

EXAMPLES

```
FASTER                  (this is usually the way to do it)
FASTER 4                (with NEWDOS, if you need four files)
FASTER 40000            (to specify a starting address)
FASTER 55000,6,50000    (start at 55000, six files, MEM=50000)
```

USING FASTER

Once activated, FASTER leaves you in BASIC. Load and run a BASIC program you want to speed up. It is possible that you will get an "Out of Memory" (OM) error with some programs. If this happens, you should look through it for the "CLEAR" statements (FASTER will tell you where they are), and modify them to CLEAR a little less string space (100-200 bytes less is usually a sufficient reduction). Then, run the program again.

It isn't necessary to run the BASIC program to completion; just run it long enough to ensure that a representative sample of its activity has been shown to FASTER. By the way, because of the analysis, your BASIC program will run somewhat slower when under the control of FASTER. Since this is only going to happen once, you need not be concerned about it. However, you won't want to (or need to) keep FASTER in memory except when doing the one-time analysis of each program.

When you feel the BASIC program has run long enough (a few minutes for an accounting program, several moves for a game, etc.), you can activate FASTER's "Option Selection Mode" (even while the BASIC program is still running) by simultaneously pressing the three keys: "567". (It's O.K. to hit <BREAK> first, but the BASIC program won't automatically resume running after you get FASTER's output.) This will suspend execution of the BASIC program and cause the menu to appear at the bottom of the screen:

C=CLEAR TABLE, E=EXIT, S=SUMMARY, D=DETAIL, V&P=VIDEO/PRINTER

The first time you use FASTER, just press "V". A list of variables, along with the number of times they were used, will be displayed. If the list fills the screen, FASTER will stop and wait for you to press <ENTER>. The very last entry in the list is the word, "END". After completing this report, FASTER identifies some program lines to be checked, then returns control to whatever program had been running. It doesn't clear the counters in the table, so you can use "567" again and again to get other readings, summary lists, or printouts. You can even let the BASIC program run a while longer and then get another analysis to compare against the first one.

OPTIONS OF FASTER

Whenever "567" is pressed, FASTER displays the available choices at the bottom of the screen:

"C" resets the counters (so do "NEW", "LOAD", "CLOAD", "CLEAR", and "RUN"). You can use "C" when you have a lengthly initialization that you don't want included in the analysis. At first, you won't use this option too often.

"E" lets you get out of "Option" mode when you didn't want to be there in the first place. It's a "WHOOPS!" command.

"S" tells FASTER to display only the variables and not the counters. The list is still sorted from "most-used" to "least-used." Summary mode is easier to read than Detail mode, and is often sufficient when using FASTER. Nothing obvious happens when you press "S"; it just sets a switch.

"D" tells FASTER to display the variables and the number of times they were used. This is the default if you don't select 'S' or 'D'. Nothing obvious happens when you press "D"; it just sets a switch.

"V" causes FASTER to immediately display its output on the video. If there is too much output to fit on the screen, FASTER pauses after every 15 lines. Press <ENTER> when you are ready to continue. If you don't have a printer, you'll want to copy the list down on paper as you go.

"P" causes FASTER to immediately display its output on the printer (and also on the video). FASTER will still pause every 15 video lines and wait for you to press <ENTER>.

'S' and 'D' leave you in FASTER's selection mode so that you can select another option. The other choices ('C', 'E', 'V', 'P') perform their functions and return control to whatever was suspended from running when you pressed "567". If you were still running your BASIC program, it will continue as though nothing had happened. If you want to select another FASTER option, just press "567" again.

OUTPUT FROM FASTER

If you followed the instructions above (loaded/activated FASTER, loaded/ran a BASIC program, pressed "567", and then pressed "V"), then you should have gotten something like this on the screen:

```
IX 1183     M$ 960     ZZ! 412     RX( 517

CHECK THESE LINES: 20 30 618
```

(The actual names and numbers will be different, of course, since every program and every run is different).

The simple variables are listed first, and the array variables (the ones followed by a left parenthesis) are listed last. This is the correct order you should use when you add your line of code to the program.

All variables are shown with their actual data types ($,%,!,#), and these data types should be included in the list you will add to the program unless you know that the defaults are correct without them. If you aren't sure, include the data type symbols as shown below.

The above example tells you two important things: the best allocation sequence for variables in this program is:

I%, M$, ZZ!, and R%(???)

and three lines (20, 30, and 618) may need to be examined when you make your speed change. Now, all you need is to learn how to use this information.

WHERE TO MAKE THE SPEEDUP CHANGES

In the simplest case, you will want to add one line of code at the very top of the program, and then save the modified program. That one line is a "DIM" statement, and it allocates the variables in the sequence FASTER showed you. In the case of the sample program above, the added line of code might be:

```
0 DIM I%, M$, ZZ!, R%(10)
```

(That's right, add Line Zero.)

You will use whatever variables FASTER displayed, not the ones shown here.

Since FASTER slows things down while its doing its analysis, you'll want to re-boot after recording its output. Then, re-load your BASIC program (with any other machine-language setups you normally use) without FASTER, and begin to make the necessary changes.

WHAT AFFECTS THE PLACEMENT OF SPEEDUP CHANGES

The placement of the "DIM" statement is very important, and the sizes of the dimensions on the array variables (the ones followed by a left parenthesis) must agree with what is already in the program. Let's take these in reverse order.

The final message from FASTER gave you a list of lines to be checked in the BASIC program. LIST or LLIST each of them, looking for these kinds of statements:

```
CLEAR DIM DEFSTR DEFINT DEFSNG DEFDBL
```

The "DIM" statements in the original program contain the sizes of the arrays. Write these down, by array name. For example:

```
600 DIM R%(35)
```

means that, when you allocate "R%(???)" in your "DIM" statement, you will replace the question marks with the number "35". If there are several numbers in the parentheses, you should duplicate all of them in your "DIM" statement:

```
610 DIM S(7,3,5)
```

means you should replace "S(???)"'s question marks with "7,3,5".

If there is a variable (letters instead of numbers) within the original DIM allocation, you should plan to use the same variable (or variables) in your allocation. However, since this can get tricky, it may be best to just leave those kinds of array variables alone (that is, don't include them in your "DIM" statement, and don't delete them from the lines they were in originally).

Now that you have the dimension values for all, or most of, the arrays, you can find out where to place your "DIM" statement.

Re-examine the lines listed by FASTER. If none of them contains "CLEAR", you can place your "DIM" statement above the original first line of the program. However, most programs have at least one "CLEAR" statement. If there is just one "CLEAR", or if all the "CLEAR" statements are in the same place, then your "DIM" statement should go right after the last statement containing a "CLEAR". If there are several "CLEAR" statements scattered through the program, you will want to place your "DIM" statement after the last one that precedes the main execution of the program. Of course, figuring this one out isn't always too easy, and this is the one potentially tricky part of using FASTER.

LOCATING THE CORRECT "CLEAR" STATEMENT

Fortunately, BASIC makes it very easy for you to find the correct answer to this puzzle. If you can't easily tell which "CLEAR" to use, just add a line of temporary code after each of them. That line should say:

```
nnn PRINT "*** AT LINE nnn ***" : STOP
```

'nnn' is the line number of the statement you're adding, and should be chosen so that it fits between the "CLEAR" and the next line of code. If there isn't any room between them, then just tack the "PRINT" and "STOP" statements at the end of the "CLEAR" statement lines themselves.

Example

```
20 GOTO 500
   ..
   ..
500 CLEAR 0 : CLEAR MEM-1000
501 print "xxx at 501 xxx" : stop
   ..
   ..
```

In this example, we added line "501" to the program.

Next, run the program. Each time it displays one of your messages, write down the displayed line number, then type "CONT" and press <ENTER>. When these messages stop appearing and the program starts performing its main functions, you'll know that the last 'nnn' is the one you want to work with.

Delete all the temporary code you just added. Then, add line 'nnn' (or add to the existing "CLEAR" line if there was no room for a line of your own). That added line or code is merely the "DIM" statement we've been wanting to add! In the example above, line 501 is probably the one to add!

```
501 DIM I%, A$, ZZ!, R%(35)
```

MAKING THE SPEEDUP CHANGE

If the newly added "DIM" statement allocates any arrarys, you must eventually delete those array allocations from other parts of the program. Since FASTER told you where all of those were, it is easy to use BASIC to edit those lines and delete the necessary information.

You will probably find that some of the arrays identified by FASTER were not allocated in any DIM statements. Such arrays should still be allocated in their suggested places in your "DIM" statement, each with a value of '10':

```
1 DIM Q(10),R%(10),S$(10)  ... etc.
```

When building your DIM statement, place the simple variables first, and the array variables last. Be sure to include all the variables shown by FASTER, and in the same sequence that they were shown. Place the DIM statement after the correct "CLEAR" statement, and then either SAVE (CSAVE) the program, or else run it (you'll want to do both, of course, and it doesn't much matter which you do first). Be sure to save it eventually, and at first you may want to save it on a different tape/disk, under a different name.

The final step is easy! just run the program and see how much faster it goes. If you're into stopwatches, time a repeatable function before you make the FASTER changes, and then time them again afterwards. Remarkable, isn't it? Such a simple change for such great results. But just think what would have been involved in trying to figure out that "DIM" statement if you didn't have FASTER!

POSSIBLE PROBLEMS

FASTER should work well and easily with most of your programs. However, there are a few situations that will require special attention:

1. An Out of Memory (OM) error occurs when using FASTER.
   FASTER requires about 1250 bytes of memory for itself, and three extra bytes of memory (for counters) for each variable referenced in your program. If your BASIC program uses most or all of memory, you can try to make some extra space just for the FASTER analysis. (You won't need this space afterwards, just the one time that you're making the analysis.) Some suggestions for getting more space temporarily are:

   * reduce the string space in the CLEAR statements
   * don't load machine language programs not absolutely needed
   * run the BASIC program through a compression program
   * make the run on a friend's larger-memory machine
   * be sure there were at least 50 bytes of non-string space available (PRINT MEM). The ROM cannot recover from insufficient numeric space.

2. FASTER conflicts with another machine-language program.
   * Check for memory location conflicts. FASTER is relocatable.
   * Try to run without the other program.

3. After adding the "DIM" statement, you get "REDIMENSIONED ARRAY" (DD) error messages.
   You allocated an array as suggested by FASTER, but didn't delete it from the statement that allocated it originally. Make that deletion now.

4. You can't figure out where to place the DIM statement.
   This shouldn't be a problem too often. Please review the instructions on "CLEAR" statements given above, and try to add the "DIM" statement at the end of one of those "CLEAR" statements, or else as a new line directly after one of them.

5. The changes didn't speed anything up. There are four likely possibilities:
   * The "DIM" is in the wrong place
   * The variables are in the wrong order (rerun FASTER and check)
   * It is faster, but you didn't time it before and after
   * The BASIC program was written correctly in the first place

   In the last case, it was most likely optimized by FASTER (or a stupendous manual effort) before you received it.

OTHER SUGGESTIONS FOR SPEEDING UP BASIC PROGRAMS

* Move all one-time initialization code to the end of the program. Just as BASIC scans the variable-name table top-to-bottom, it scans for BASIC statements from top-to-bottom. If the initialization is all at the top, it'll have to be bypassed over and over again during the execution of the main portion of the program.

* Place the most-used routines at the beginning of the program. Like the "variable allocation" problem that FASTER solves, this one is easy to suggest, but usually difficult to implement. If you didn't write the BASIC program in the first place, it's probably not practical to even try it.

* Use DEFINT for as many of your numeric variables as possible

STEP-BY-STEP SAMPLE RUN AND USE OF FASTER

This section will lead you through a complete session using "FASTER". A very simple BASIC program is used for purposes of illustration. The program does contain a "CLEAR" statement and a "DIM" statement to help you see how they affect the changes you should make when using FASTER.

Initialization in Disk Systems

(If you're running LEVEL II, skip this paragraph.)

1. Power up with your normal DOS.
2. Load any machine language programs required by the BASIC program you plan to analyse. If memory may be a constraint, don't load any optional programs such as lower-case drivers or spoolers.

3. Mount a disk containing FASTER and type: FASTER

Now skip down to step #7 in the "Level II" sample run below.

LEVEL II INITIALIZATION

1. Power on.
2. Answer the Memory Size as you normally do.
3. Load any System programs you need. Don't load any optional routines unless you're sure there will be sufficient memory when FASTER is resident.

4. Place the FASTER tape in the tape recorder and press "PLAY".
5. Load and activate FASTER:

    1. Type "SYSTEM" (no quotes) and press <ENTER>
    2. Type "FASTER" and press <ENTER>
    3. When loading completes, press slash ("/") and then <ENTER>.
    4. Rewind and remove the tape.

6. CLOAD the BASIC program to be analysed.

7. RUN your BASIC program and exercise its functions.
8. Let the program perform its main functions for a couple of minutes (the length of time needed varies tremendously: it may be 30 seconds, or several minutes in extreme cases: if you aren't sure, get several FASTER readings by pressing "567" and then "V" every 30 - 60 seconds, until the list of variables keeps coming out in the same sequence).

9. Press "567" to gain access to FASTER's OPTION menu.
10. Press "V" or "P" to get the detail analysis.
11. If you're not sure the BASIC program ran long enough, let it continue (it does so automatically if you didn't hit <BREAK>), then repeat steps 9-10.

12. In the sample program shown below, the most frequently used variables, and the lines to be checked were:

    L, K, I, J, N, M1, M, A(, C$(
    CHECK THESE LINES: 30, 40

    (The detail counts aren't shown because they'll probably be different from the ones you get: I pressed "567" at a different moment than you will.)

13. Write down the information if you don't have a printer.
14. Re-boot.
15. Set MEM SIZE normally, load any needed SYSTEM programs, and LOAD or CLOAD your BASIC program. DO NOT LOAD FASTER this time.

16. Examine all the lines that FASTER asked you to check. If any of them contain "DIM" statements, write down the variables and their dimensions. In the example given below, they are:

    40 DIM C$(17), A(255)

17. If there are no "CLEAR" statements in the program, you can insert your speedup code above the first line of the program. In our example, line 30 contains a "CLEAR", so the new line of code will be line 31.

18. Delete all array allocations from the original program. In our example, this means that line 40 can be deleted in its entirety.

19. If FASTER identified any array variables that you can't find in DIM statements, you should assign them a default dimension of "10". Our example doesn't have any of these.

20. The changes should be complete and the program ready for testing. In our example, we added line 31 and deleted line 40, and the result is "DEMO2".

21. RUN the new program and look for two things:

    1. Are there any error messages due to "Redimensioned Arrays" ("DD" in Level II)? If so, remove the DIM statements from the original program, leaving only the one(s) you added.

    2. Does it run faster? If your DIM statement is right after the correct "CLEAR", or is the first line of the program if there aren't any "CLEAR" statements, it should run quite a bit faster.

22. An additional change that would probably further benefit this sample program would be to specify INTEGER arithmetic:

    31 DEFINT A-Z : DIM L, K, I, J, N, M1, M, A(255), C$(17)

Even in this trival example, use of FASTER produced a 12% speed improvement in the " Thinking" phase. (More complex programs usually show more improvement.) Then, by adding a "DEFINT", a further speedup was obtained, but that had nothing to do with FASTER.

This completes the sample run. Now, if you haven't already done so, analyse some programs in your own library, and enjoy their "FASTER" operation.